glob

wiki.tcl-lang.org/page/glob

glob , a built-in Tcl command, matches files by name.

See Also

dir listing
A drop-in replacement for glob that adds another type specifier, +hidden, to include hidden files in all results.

Directory recursion

glob forwards compatibility
how to to override the older [glob] command.

glob and VFS
issues with VFS

Synopsis

glob ?*switches...*? *pattern* ?*pattern ...*?

Supported *switches* are:

-directory *directory*

-join

-nocomplain

-path *pathPrefix*

-tails

-types *typeList*

--

Documentation

official reference

Description

glob lists all the directory entries whose names match a given *pattern*, optionally filtering them by type.

The following two cases are errors:

- The result set is empty.
- The requested listing can't be completed, e.g. when permissions are insufficient.

-nocomplain can be used to squelch the error in the first case, except that currently, there is a bug that causes glob to also squelch a failure to obtain the type of a file in order to limit the results to files of a certain type. ycl dir listing is similar to glob, but works around this issue.

glob should have been designed such that it returned the empty string when the result list would be empty, making -nocomplain unnecessary, but for the sake of backward-compatibility, in the current Tcl release glob still returns an error if the result list would be empty.

glob raises an exception; the -nocomplain option

If no files or directories were found, the command throws an error, unless the **-nocomplain** switch is given. This models the behavior of Unix's **ls**:

```
$ ls not_existing
ls: not_existing: No such file or directory
$ echo $?
1
```

PYK 2016-11-16: Except that it doesn't model the behaviour of ls in other cases, such as when it operates on an existing directory that has no contents.

Use -- to separate switches from arguments

Take care to use -- to signify the end of switches whenever *patterns* contain arbitrary characters or might get a preceding dash through substitution.

The -types option and hidden files

-types can be used to filter the results to certain types of files:

```
glob -types hidden *
glob -types d *
```

There are cross-OS issues with this the handling of hidden files. On unix and OS X, glob -types hidden * returns all files beginning with a period (.), including . and ... On Windows and Classic Mac OS, this returns files that have their hidden attribute set (and (at least on Windows) *does not* list files with names beginning with a period, except if they have the hidden attribute set). On Windows, glob * returns files beginning with ., while on *nix systems, it does not.

The following script gets all files and directories (hidden or otherwise, plus includes . and ..) on *nix, but can return duplicates on Windows:

```
glob -directory $dir {.*,*}
```

To get a list of all contents of a directory that aren't themselves directories, including broken symbolic links (lsort -unique is used in case an item gets listed by both invocations of glob):

```
set itemlist [
    concat [
        glob -nocomplain -directory $dir -types hidden *] [
        glob -nocomplain -directory $dir *]]
#usually . and .. aren't desired
set onlyfiles [lmap item [lsort -unique $itemlist] {
    expr {![file isdirectory $item] && [file tail $item] ni {. ..}
        ? $item
        : [continue]
    }
}]
```

[name redacted]: I rewrote the script some.

I left the subcondition [file tail $item] ni {. ..} in, even though . and .. *are* directories and thus get filtered out by ![file isdirectory $item] alone, i.e. this invocation ought to do:

```
expr {![file isdirectory $item] ? $item : [continue] }
```

PYK 2014-08-13: edited the example above to use [file tail $item] instead of $item, as the -directory option causes glob to return paths rather than simple file names.

AMG: The following should work mostly like [glob], except it returns both hidden and non-hidden files and directories matching all the other criteria. If hidden is explicitly included in the -types argument, only hidden results are returned. . and .. are excluded from the result in all cases. Also, -nocomplain is permanently engaged, since life is easier that way.

```
proc globAll {args} {
    set opt -nocomplain
    set types {}
    while {[llength $args] && [string index [lindex $args 0] 0] eq "-"} {
        set args [lassign $args arg]
        switch [set arg [tcl::prefix match {
            -directory -join -nocomplain -path -tails -types --
        } $arg]] {
        -join - -tails {
            lappend opt $arg
        } -directory - -path {
            lappend opt $arg [lindex $args 0]
            set args [lrange $args 1 end]
        } -types {
            set args [lassign $args types]
        } -- {
            break
        }}
    }
    concat [lmap e [glob {*}$opt -types [list {*}$types hidden] -- {*}$args] {
        expr {[file tail $e] in {. ..} ? [continue] : $e}
    }] [if {"hidden" ni $types} {glob {*}$opt -types $types -- {*}$args}]
}
```

The -directory and -path options

(Note that these are mutually exclusive. Also note that -directory can be abbreviated to -dir.)

Let's say I want to find all files in a directory, then I use (ignoring hidden files for the moment)

```
glob -dir $dir *
```

Now, what I want all files in that directory whose names start with $nametail (where I have no idea what characters are in $nametail). I could try:

```
glob -dir $dir ${nametail}*
```

but that would fail (either with an error or not the expected result) if nametail contains characters like {}. Therefore I must use *-path*, like this:

```
glob -path $dir/$nametail *
# or even (see below)
glob -join -path $dir $nametail *
```

One thing to be aware of is that -dir {} is the same as -dir .. Hopefully this wart will be removed in the future.

The -join option

When to use *-join*?

If I want to find all *.tcl files in any subdirectory of a known directory, I could try this:

```
glob -dir $dir */*.tcl
```

but that won't work on MacOS. So, use -join:

```
glob -join -dir $dir * *.tcl
```

Trailing /

In a recent email thread on the Modules mailing list, a developer reported a change he found when beginning to use Tcl 8.5:

```
$ tclsh8.4
% glob /home/
/home
% glob /home
/home
^D
$ tclsh8.5
% glob /home/
/home/
% glob /home
/home
^D
```

Turns out that the modules package was dependent on having the path returned without the trailing /, and when 8.5 started returning some paths with the slash, certain behaviors changed.

From a purely technical point of view, the two forms of the name are considered equivalent by the operating systems with which I am familar (Ultrix, Solaris, etc.). I don't know if there are differences for variants or not.

[name redacted]: In Tcl 8.6, this seems to be changed *again*: no trailing slashes are in the result.

In any case, file normalize removes any trailing slashes present:

```
string equal [file nor [glob /Tcl/] [file nor [glob /Tcl]
# => (should be 1 on all versions)
string equal file nor /foo/ file nor /foo
# => 1
```

APN I don't see 8.6 having changed again. At least not on Windows. If the pattern include a trailing slash, the return values do too (for dirs). Otherwise they don't.

```
% glob /temp/fo*
C:/temp/foo C:/temp/foodir
% glob /temp/fo*/
C:/temp/foodir/
```

glob does not sort

The glob command does not return sorted results, so to emulate csh one might have to use lsort on the file list (note the various settings of the lsort command: especially differences between -ascii and -dictionary sorting, the option to choose increasing or decreasing order, the option to ignore case difference, and the option to provide an adhoc comparison command):

```
set files [lsort [glob *]] ;# will probably suffice in most cases
set files [lsort -dictionary -decreasing -nocase [glob *]]
```

Negated patterns

LV: does anyone have some Tcl code for handling negated patterns? The ksh glob function allows me to do things like !(o*) or ab[!b-z] and I would like to be able to do similar things.

[name redacted]: This one is still open, though one might possibly glob two lists, one with all files and one with the pattern-to-be-negated, and ldiff the second from the first. There may be issues with this that need to be investigated, though.

[name redacted]: The following command is intended to *start* a discussion, not to be a solution (among other things, it does nothing for the ab[!b-z] case).

```
# untested code
proc negatedGlob pattern {
    set allfiles [glob -nocomplain *]
    set matchedfiles [glob -nocomplain $pattern]
    ldiff $allfiles $matchedfiles
}
```

Use of braces

One might consider at least one of the behaviours of glob as a wart. glob uses the brace ({}) characters as special notation, even though the Tcl parser *ALSO* uses these characters. The result is that one needs to learn, from the beginning, that one needs to quote these characters when using them in glob arguments. glob also uses the [and] characters and those need quoted as well.

Vince: fortunately with the use of '-dir' or '-path' flags to glob the need for quoting is pretty much gone in Tcl 8.3 or newer.

Filename Encodings

PYK 2017-08-19: glob presumes that file names are encoded in the system encoding. That presumption, along with the fact that utf-8 conversions accept any sequence of bytes, including invalid utf-8, means that a round-trip from utf-8 and back again corrupts

the filename. This can lead to cases where other file command fail when handed a filename produced by glob. open might produce the error no such file or directory, and file exists might return false.

iso8859-1 is an 8-bit encoding whose code points either correspond to the same code points in Unicode or are undefined, so a round-trip conversion will always result in the original string. It can be used to ensure that file names produced by glob remain valid for other commands that interact with the filesystem:

```
set encoding_save [encoding system]
try {
    encoding system iso8859-1
    set files [glob *]
} finally {
    encoding system $encoding_save
}

# and later ...
set encoding_save [encoding system]
set file [lindex $files 0]
try {
    encoding system iso8859-1
    set chan [open $file]
} finally {
    encoding system $encoding_save
}
```

See also, glob, encoding system and encoding-free filesystems

Recursive glob

Recursive glob: This lightly tested procedure follows subdirectories (not sure what happens with links) and returns all non-directories in alphabetic order:

```
proc glob-r {{dir .}} {
    set res {}
    foreach i [lsort [glob -nocomplain -dir $dir *]] {
        if {[file type $i] eq {directory}} {
            eval lappend res [glob-r $i]
        } else {
            lappend res $i
        }
    }
    set res
} ;# RS
```

(Verified works with 8.6.)

JH 2006-05-16: With this variant that allows for optional patterns of interest, like glob-r C:/Tcl *.gif *.ps:

```
proc glob-r {{dir .} args} {
    set res {}
    foreach i [lsort [glob -nocomplain -dir $dir *]] {
        if {[file isdirectory $i]} {
            eval [list lappend res] [eval [linsert $args 0 glob-r $i]]
        } else {
            if {[llength $args]} {
                foreach arg $args {
                    if {[string match $arg $i]} {
                        lappend res $i
                        break
                    }
                }
            } else {
                lappend res $i
            }
        }
    }
    return $res
} ;# JH
```

(Invocation without arguments verified works with 8.6.)

MG 2004-05-20: I couldn't get the above code to work, so wrote my own version of it. The only oddity about it is that paths (sometimes) start ././, rather than just ./ - I really don't know why. But using file exists on those files still works, so it doesn't really seem to matter. (It also uses info level 0 to find out its own name when it calls itself recursively. This is simply because I just found out how you do that :) Replacing [lindex [info level 0] 0] with globRec (or whatever you call the procedure) will speed it up a fair bit. *Changed quickly to add -- into the glob calls, so $dir can't be mistaken as a switch, as suggested above*. Takes an optional second argument to specify which types of file to glob, for by default all mentioned on the manpage apart from directories.

```
proc globRec {{dir .} {types {b c f l p s}}} {
    set files [glob -nocomplain -types $types -dir $dir -- *]
    foreach x [glob -nocomplain -types {d} -dir $dir -- *] {
        set files [
            concat $files [[lindex [info level 0] 0] [file join $dir $x]]]
    }
    return [lsort $files]
} ;# globRec
```

(Invocation without arguments verified works with 8.6.)

TC 2004-11-23: I couldn't get either of the above functions to do what I wanted, which was to recursively list all directories and use a filespec. My Tcl hack (lightly tested on 8.3 and 8.5) of MG's code, which seems to do all his was intended to do also. Modified to also eliminate leading ./s. Modified to support filenames with spaces in them.

Note: you may need to replace file join $dir $x with simply $x. I had to.

```
proc globRec {{dir .} {filespec *} {types {b c f l p s}}} {
    set files [glob -nocomplain -types $types -dir $dir -- $filespec]
    foreach x [glob -nocomplain -types {d} -dir $dir -- *] {
        set files [concat $files [globRec [file join $dir $x] $filespec $types]]
    }
    set filelist {}
    foreach x $files {
        while {[string range $x 0 1] eq {./}} {
            regsub ./ $x {} x
        }
        lappend filelist $x
    }
    return $filelist;
};# globRec
```

(Invocation without arguments verified works with 8.6.)

---

jys: I wanted a recursive glob that would take the same options as glob, so I wrote this wrapper. I don't know if it works 100% with all options though, and it currently gets tripped up by glob spitting the dummy when it can't look inside a directory (generally 'cause of permissions).

```
proc rglob {args} {
    # This code requires -join and -nocomplain. We also remove the -types option,
and then manually apply a filter at the end.
    set args [linsert $args 0 -nocomplain -join]
    if {[set types_index [lsearch -exact $args -types]] != -1} {
        set types [lindex $args [expr {$types_index+1}]]
        set args [lreplace $args $types_index [expr {$types_index+1}]]
    }

    # Get inital matches.
    set matches [glob {*}$args]

    # Keep adding * wildcards to search through subdirectories until there are no
more levels of directories to search.
    while {[llength [glob -types d {*}[lreplace $args end end *]]] > 0} {
        set args [linsert $args end-1 *]
        lappend matches {*}[glob {*}$args]
    }

    # Filter matches with -types option.
    if {[info exists types]} {
        set matches [glob -nocomplain -types $types {*}$matches]}
    return $matches
}
```

(Hangs Tcl 8.6 (made no attempt to find cause of problem).)

findfile

ES: **findfile:** This procedure was inspired by glob-r above and is customized to quickly find the first instance of a file in a given path.

```
proc findfile { dir fname } {
    if {
        [llength [set x [glob -nocomplain -dir $dir $fname]]]
    } {
        return [lindex $x 0]
    } else {
        foreach i [glob -nocomplain -type d -dir $dir *] {
            if {
                $i != $dir &&
                [llength [set x [findfile $i $fname]]]
            } {
                return $x
            }
        }
    }
}
```

(Verified works with 8.6.)

listTree

Here is a proc that will list the directory tree under any given root, giving the full path name of all descendants. It embeds glob in a recursive call; a directory with no descendants is the base case.

```
proc listTree {rootdir_} {
# Precondition: rootdir_ is valid path
    set currentnodes [glob -nocomplain -directory $rootdir_ -types d *]
    if {[llength $currentnodes] <= 0} {
    # Base case: the current dir is a leaf, write out the leaf
        puts $rootdir_
        return
    } else {
                # write out the current node
        puts $rootdir_
                # Recurse over all dirs at this level
        foreach node $currentnodes {
            listTree $node
        }
    }
}

listTree [pwd]
```

(Verified works with 8.6 (and is surprisingly quick).)

(Original command author): That this isn't a Tcl built-in (or if it is, why is it is such a well-kept secret) strikes me as odd.

I remember reading something about a pre-defined limit on depth of recursion in Tcl, so this code is not guaranteed for arbitrarily deep file structures.

Bob

pathentry

MSW: pathentry: Entry widget/binding set using glob to offer quick auto-completion of paths

The DKF star

GPS: In the Tcl'ers chat this elegant solution was created by DKF:

```
proc * args {glob -nocomplain *[join $args {}]}
```

CMCc: Now try the same kind of composition and extension under /bin/sh :)

MJ: Shouldn't this be:

```
proc * args {glob -nocomplain *\{[join $args ,]\}}
```

Because DKF's version doesn't do what I expect it to do with multiple args (maybe my expectation is wrong)

Unglobbing

Sarnold: Sometimes glob-like patterns are expected to be passed to some commands. This may lead to bugs, for example, in array unset gotcha , comp.lang.tcl, 2007-09-13:

```
set bar {x[y]}
set foo($bar) yes
array unset foo $bar
parray foo => foo(x[y]) still exists
```

What's the cure, Doctor? **Unglob it!**

```
proc unglob {x} {
    string map {* \\* ? \\? [ \\[ ] \\]} $x
}
```

AMG: [glob] pattern syntax is a superset of string match pattern syntax. [glob] supports {a,b,...} syntax, which is not supported by [string match] or any other Tcl commands that I know of. [glob] also treats patterns starting with ~ as referring to the current or a named user's home directory. To accommodate, [unglob] needs to prefix braces and (initial) tildes with backslashes.

```
proc unglob {x} {
    regsub -all {^~|[]{}*?\\]} $x {\\&}
}
```

[glob] has special handling for entries starting with a period (.). However, this does not affect pattern syntax and therefore does not impact [unglob].

Windows drive globbing

kostix 2004-09-07: There's one trick with **glob** on Windows:

```
glob -dir c: *
```

will list contents of *the current working directory on drive C:,* not the contents of the root folder on that drive.

To get the latter, add trailing slash to the drive letter:

```
glob -dir c:/ *
```

The described behaviour is consistent with what is seen in **cmd.exe** command shell.

Thanks PT for explaining this issue.

Zipguy 2013-02-07 - Thanks kostix for pointing this out. I do use windows (and strongly love/hate it). I found this very interesting so I tried it. I was running a program called ML - Heavily Modified & Improved using tclkit-8.5.8.exe, in which mine was the current directory, so I opened a console window and typed in:

```
(mine) 3 % glob -dir d: *
```

and what I got was:

```
d:action-demo.tcl d:action.tcl d:awf.kit d:awfgreat.kit d:awf_cfg.ml
d:bindb.bat d:bindb.kit d:bindb.tcl d:bindb.vfs d:bindcbk.tcl d:clipr.kit
d:clipr.vfs d:colorp.dat d:colorp.kit d:dler.kit d:dler.vfs d:eff-font.txt
d:efftcl.kit d:efftcl.vfs d:ezsdx.bat d:ezsdx.kit d:ezsdxc.dat d:ezsdxgreat.kit
d:ezsdxh.dat d:help.dat d:icons d:links.txt d:ml.bat d:ml.kit d:ml.txt
d:mlalpha.kit d:mlgreat.bat d:mlgreat.kit d:mlgreatold.kit d:ml_cfg.ml
{d:ml_cfg.ml.$$$} d:output.html d:output.tcl d:output2.html d:panedwindow.tcl
d:progress.tcl d:progressbar.txt d:res.tcl d:sbf.tcl d:scrollb.tcl d:sdx.kit
d:sdx.vfs d:sdxnew.kit d:sdxold.kit d:t.kit d:t.tcl d:t.vfs d:tclkit-8.5.8.exe
d:tclkit-8.5.9.exe d:tgood.kit d:timeplay d:timeplay.tcl d:timeplay.tcll
d:Tk_coding_styles.txt d:widget d:wsf.kit d:wsf.vfs d:wsfold.kit d:wsfold.tcl
d:wsfold.vfs d:wsforig.tcl d:XSCLOCK.TCL d:yacm.ini d:yacm.kit d:yacm.txt
d:yacm2.kit d:yacmgood.kit d:yacmo.kit d:ypedit.tcl d:zipper.tcl
```

which did surprise me.

It did come back with all of these files (which was expected), which were on the proper drive, namely d:, in the proper directory, namely mine (or more precisely "D:\dl\dcl\mine\"), but, it did prefix all of them with the "d:" (which was unexpected). For example, "d:tclkit-8.5.9.exe" which is there and "{d:ml_cfg.ml.$$$}" which it did enclose in curly brackets, which I do understand (in case they have spaces or special characters in them).

Anyway, I decided to try a more complex version, which would list the directory above the current one (which is kind of dangerous if it doesn't exist, unknown) like this:

```
(mine) 5 % glob -dir d:../ *
```

and (lo and behold,) it did produce this:

```
d:../caps d:../desktop d:../ezsdx d:../ezsdx.kit d:../ezsdxc.dat
d:../ezsdxgreat.kit
d:../ezsdxh.dat d:../fd d:../freepdf d:../freepdflogoweeebxd.jpg d:../freewrap6.2
d:../irfanview_screenshot_1.jpg d:../irfanview_screenshot_2.jpg
d:../irfanview_step1.jpg
d:../irfanview_step2.jpg d:../irfanview_step3.jpg d:../jasspa-mechm-ms-win32-
20091011.zip
d:../mine d:../ml d:../ml.kit d:../ml124d_procw_colors.jpg d:../ml124d_screen.jpg
d:../ml124d_screen_rightclick_meni.jpg d:../ml124d_status_line.jpg
d:../ml124d_toolbar.jpg
d:../mlgreat.kit d:../mlgreatold.kit d:../sdx.kit d:../sdxe.dat d:../sdxnew.kit
d:../sdxold.kit
d:../setup-jasspame-20091011.exe d:../stock d:../t.kit d:../tclkit-8.5.8-win32-
x86_64.exe
d:../tclkit852.exe d:../tgood.kit d:../tgoodold.kit d:../viewIcons.kit
```

which did work but provided a prefix of "d:../" rather than the previous command which provided a prefix of "d:" which does make sense also.

```
(mine) 7 % glob -dir d:../../ *
```

and this

```
d:../../@tcl d:../../bas d:../../browser d:../../dcl d:../../dler
d:../../editor d:../../flv d:../../fonts d:../../games d:../../good
d:../../html d:../../img d:../../mid d:../../misc d:../../nc d:../../new
d:../../new2 d:../../new3 d:../../new4 d:../../new5 d:../../zip
```

But more importantly, should I write a proc to strip out the "d:"'s (or "d:../"'s) in front of them, or use a better undocumented glob option?

Yes, I did read (and re-read) all about glob for "ActiveState ActiveTcl 8.4.19.5.294332" (which was produced in Feb 11, 2011) which may mean it has been documented in a later version.

MG Add the -tails option to glob.

Stale links

RS:

Stale links: *glob* in older (pre Tcl 8.4a4) versions of Tcl has a funny behavior (seen on Solaris 5.5.1) with symbolic links that don't point to an existing file:

```
% exec ln -s /does/not/exist stalelink
% glob stalelink
no files matched glob pattern "stalelink"
% glob stalelink*
stalelink
```

If the pattern is not wildcarded, it tries to follow the link; otherwise it just returns the link's name itself. Hmm...

Tilde paths

EB: The changes in Tcl/Tk 8.4.9 shows that *tilde paths are not returned specially by glob*, but shouldn't that have been done to avoid a security problem with ~ expansion ? e.g:

```
foreach file [glob *] {
    file delete -force $file
}
```

where glob returns a file beginning with a ~ ?

Vince writes: for historical reasons the code above is considered wrong/buggy. You should write file delete -force ./$file. This is perhaps something that should be changed in the future, but it might well break some old code. There is even a bug report or patch somewhere on sf about this, I think (probably closed now).

Interpreter crash

MHo 2015-10-15: Oops, what's this:

```
% glob -path c:/winwdows/*.exe
alloc: invalid block: 10018A34: 24 e8
```

The Tclsh (8.6.4, ActiveState) terminates after this...

AMG: Is "winwdows" intentionally part of your bug report or just a typo in this wiki page?

Directory separator

Even though the proper directory separator character on Windows is \, the standard directory separator in Tcl is '/', and all Tcl file manipulation commands, including exec, understand it.

Mixing -path and -directory

AMG: [glob] forbids mixing -path and -directory, but I think the two can be useful in combination with -tails. The idea is to use -directory to specify the prefix to be stripped off by -tails, plus to use -path to specify a prefix to the glob pattern that will not be stripped off by -tails and is protected from glob metacharacter interpretation. Otherwise, it's necessary to do one of the following:

- Put everything into -directory then re-add some portion of the path to the return value
- Leave out -tails and strip some portion of the path from the return value
- Apply "unglob" to glob path prefix

Why not allow -path and -directory together? Was it because there wasn't a good use case for having both at once? Well, suppose you want to list some subdirectories of the Linux kernel source tree:

```
foreach subdir {Documentation LICENSES arch block} {
    glob -directory /usr/src/linux -path $subdir -tails *
}
```

If it weren't for this restriction, the above [glob] calls would return names like "Documentation/ABI" and "LICENSES/deprecated" and "arch/Kconfig". Here, it's possible to instead say:

```
foreach subdir {Documentation LICENSES arch block} {
    glob -directory /usr/src/linux -tails [file join $subdir *]
}
```

Or:

```
foreach subdir {Documentation LICENSES arch block} {
    glob -directory /usr/src/linux -tails -join $subdir *
}
```

But if the subdirectories had glob metacharacters (including braces and leading tilde), they would need to be backslash-quoted. -path avoids the need to quote.

For a practical application of this feature, consider a program that needs to copy one directory tree to another while applying some kind of transformation along the way. If it needs to preserve the hierarchy relative to the input base and transplant it all to the output base, then having -tails strip off the input base (argument to -directory) but not the subdirectory structure (argument to -path) makes perfect sense.

Tcl syntax	Arts and Crafts of Tcl-Tk Programming	Category Command	Category File

Updated 2023-12-02 18:48:40